Nicht für den Prüfling bestimmt

I Erläuterungen

Voraussetzungen gemäß KCBG und Abiturerlass BG in der für den Abiturjahrgang geltenden Fassung

Standardbezug
Die nachfolgend ausgewiesenen Kompetenzbereiche sind für die Bearbeitung der jeweiligen Aufgabe besonders bedeutsam. Darüber hinaus können weitere, hier nicht ausgewiesene Kompetenzbereiche für die Bearbeitung der Aufgabe nachrangig bedeutsam sein, zumal die Kompetenzbereiche in engem Bezug zueinanderstehen. Die Operationalisierung des Bezugs zu den Kompetenzbereichen des Standardbezugs erfolgt in Abschnitt II.

Aufgabe	Kompetenzbereiche				
	K1	K2	K3	K4	K5
1.1		X	X		
1.2				X	
1.3			X		
1.4		X		X	
1.5				X	
1.6.1	X				X
1.6.2		X			
1.6.3				X	
2.1	X				
2.2		X			
2.3.1			X	X	
2.3.2				X	
2.3.3				X	
2.3.4			X	X	
2.3.5			X	X	
2.4		X	X		

Inhaltlicher Bezug
Die nachfolgend ausgewiesenen Themenfelder sind die wesentliche inhaltliche Grundlage für die vorliegenden Aufgaben. Darüber hinaus können weitere, hier nicht explizit ausgewiesene Themenfelder für die Bearbeitung nachrangig bedeutsam sein.
Q1: Objektorientierte Softwareentwicklung
Q2: Datenbanksysteme
verbindliche Themenfelder: Objektorientierte Modellierung (Q1.1), Implementierung von Klassen und Assoziationen (Q1.2), Suchen und Sortieren (Q1.3), Konzeptionelle und logische Modellierung einer Datenbank (Q2.1), Datenabfrage und Datenmanipulation mit SQL (Q2.2)

II Lösungshinweise

In den nachfolgenden Lösungshinweisen sind alle wesentlichen Gesichtspunkte, die bei der Bearbeitung der einzelnen Aufgaben zu berücksichtigen sind, konkret genannt und diejenigen Lösungswege aufgezeigt, welche die Prüflinge erfahrungsgemäß einschlagen werden. Selbstverständlich sind jedoch Lösungswege, die von den vorgegebenen abweichen, aber als gleichwertig betrachtet werden können, ebenso zu akzeptieren.

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.1	entwickeln, zeichnen : Spieler name = Christian Müller : Karte zahlFigur = Dame zeile = 1 wert = 10 : Symbol bezeichnung = Herz sortSymbol = 2 : Karte zahlFigur = Dame zeile = 1 wert = 10 : Symbol bezeichnung = Karo sortSymbol = 1 : Karte zahlFigur = Joker zeile = 11 wert = 50 : Symbol bezeichnung = Joker sortSymbol = 0 : Karte zahlFigur = 7 zeile = 7 wert = 5 : Symbol bezeichnung = Kreuz sortSymbol = 4 entwickeln zeichnen	 3	2	1

1.2	implementieren
	`private void mischen() {`
	`  Random rand = new Random();`
	`  Kartensatz gemischteKarten = new`
	`    Kartensatz(ANZAHL_KARTEN);`
	`  List<Karte> ungemischt = kartenspiel.getKarten();`
	`  while (ungemischt.size() > 0) {`
	`    Karte tmpKarte =`
	`      ungemischt.remove(rand.nextInt(ungemischt.size()));`
	`    gemischteKarten.hinzufuegenKarte(tmpKarte);`
	`  }`
	`  this.kartenspiel = gemischteKarten;`
	`}`
	Hinweis: Die Implementierung einer anderen Methode, die eine Ermittlung per Zufall beinhaltet, ist als gleichwertig zu bewerten.
			4	

Aufg.	erwartete Leistungen	BE I	BE II	BE III
1.3	entwickeln, zeichnen stapel : List<Karte> aktuelleHandkarten : List<Karte> sp : Spieler haufen : List<Karte> : CanastaManager ausfuehrenSpielzug(sp:Spieler) holeVomHaufen() remove(0) { k } { k } fuegeSortiertEin(k) add(index, k) waehleKarte() { pos } entferneKarte(pos) remove(pos) { karte } { karte } legeAufStapel(karte) add(karte)			
	entwickeln		2	2
	zeichnen	4		

Aufg.	erwartete Leistungen	BE		
		I	II	III

1.4	überführen, implementieren
	`public class Spieler {`
	`  private List<Karte> aktuelleHandkarten;`
	`  private String name;`
	
	`  public Spieler(String name) {`
	`    this.name = name;`
	`    this.aktuelleHandkarten = new List<>();`
	`  }`
	
	`  public Karte entferneKarte(int pos) {`
	`    return this.aktuelleHandkarten.remove(pos);`
	`  }`
	`  public void fuegeSortiertEin(Karte k) {`
	`    int i = 0;`
	`    while (i < this.aktuelleHandkarten.size() &&`
	`      this. aktuelleHandkarten.get(i).compareTo(k)) < 0) {`
	`      i++;`
	`    }`
	`    aktuelleHandkarten.add(i, k);`
	`  }`
	`}`
	
	`public class Karte implements Comparable<Karte> {`
	`  private String zahlFigur;`
	`  private int zeile;`
	`  private int wert;`
	`  private Symbol symbol;`
	
	`  public Karte(String zahlFigur, int zeile, Symbol symbol,`
	`    int wert) {`
	`    this.zahlFigur = zahlFigur;`
	`    this.zeile = zeile;`
	`    this.symbol = symbol;`
	`    this.wert = wert;`
	`  }`
	`  private int vergleicheSymbol(Karte andereKarte) {`
	`    int code;`
	`    if (andereKarte.getSymbol().getSortSymbol() ==`
	`      this.getSymbol().getSortSymbol()) {`
	`      code = 0;`
	`    } else if (andereKarte.getSymbol().getSortSymbol() <`
	`      this.getSymbol().getSortSymbol()) {`
	`      code = -1;`
	`    } else {`
	`      code = 1;`
	`    }`
	`    return code;`
	`  }`			

<table>
<tr><th rowspan="2">Aufg.</th><th rowspan="2">erwartete Leistungen</th><th colspan="3">BE</th></tr>
<tr><th>I</th><th>II</th><th>III</th></tr>
<tr><td></td><td><pre>
  public int compareTo(Karte andereKarte) {
    int code;
    if (andereKarte.getZeile() == this.getZeile())
      return vergleicheSymbol(andereKarte);
    }
    if (andereKarte.getZeile() < this.getZeile())
      code = 1;
    else
      code = -1;
    return code;
  }
  public boolean isJoker () {
    return (zahlFigur.equals("Joker") ||
                                zahlFigur.equals("2"));
  }
}
</pre>
überführen
implementieren</td><td>4</td><td>6</td><td>4</td></tr>
<tr><td>1.5</td><td>implementieren<pre>
  public int ermittlePunkte() {
    int punkte = 0;
    for (int i = 0; i < flaeche.length; i++) {
      punkte = punkte + punkteZeile(i);
    }
    return punkte;
  }
  private int punkteZeile(int zeile) {
    int punkte = 0;
    int kartenzahl;
    int anzahlJoker = 0;
    for (kartenzahl = 1; kartenzahl <=
      flaeche[zeile].length; kartenzahl++) {
        if (flaeche[zeile][kartenzahl - 1] == null)
          break;
      punkte = punkte +
               flaeche[zeile][kartenzahl - 1].getWert();
      if (flaeche[zeile][kartenzahl - 1].isJoker())
        anzahlJoker++;
    }
    if (kartenzahl == 7)
      punkte = punkte + punkteCanasta(anzahlJoker);
    return punkte;
  }
  private int punkteCanasta(int anzahlJoker) {
    int punkte = 300;
    if (anzahlJoker == 0)
      punkte = 500;
    else if (anzahlJoker == 7)
      punkte = 1000;
    return punkte;
  }
</pre></td><td></td><td>3</td><td>5</td></tr>
</table>

<table>
<tr><th rowspan="2">Aufg.</th><th rowspan="2">erwartete Leistungen</th><th colspan="3">BE</th></tr>
<tr><th>I</th><th>II</th><th>III</th></tr>
<tr><td>1.6.1</td><td>erklären
Die Vorgehensweise des vorliegenden Sortieralgorithmus wird maßgeblich durch die Methode <code>quicksort()</code> festgelegt, die in der main-Methode aufgerufen wird und die das sortierte Zahlenarray zurückgibt. Ihr wird das in der main-Methode bereitgestellte Zahlenarray sowie sein erster Index <code>start</code> und sein letzter Index <code>ende</code> übergeben. Die dargestellte Methode <code>quicksort()</code> ist eine rekursive Methode. Rekursive Methoden rufen sich immer wieder selbst auf, bis ein Rekursionsanker erreicht ist, der diesen Prozess beendet. Immer, wenn <code>start < ende</code> wahr ist, ist das Ende der Rekursion noch nicht erreicht, und die Methode <code>teileArray()</code> wird aufgerufen. Sie liefert den Index <code>q</code>, dessen zugehöriges Element (Pivotelement) bereits an seinen Platz im später fertig sortierten Array getauscht worden ist. Nun muss in den Rekursionsschritten, also in den zwei darstellten Aufrufen der Methode <code>quicksort()</code>, im Teilarray vor und im Teilarray hinter dem Index <code>q</code> weiter sortiert werden, was aber nur durchgeführt wird, wenn es noch einen unsortierten Bereich des Arrays gibt, also dessen erster Index vor seinem letzten Index liegt. Die dargestellte Ausgabe von <code>q</code> erfolgt zu Testzwecken und gehört normalerweise nicht zur Methode <code>quicksort()</code>.</td><td></td><td>2</td><td>3</td></tr>
<tr><td>1.6.2</td><td>bestimmen, angeben
<code>0<2?</code> ja
  <code>0<3?</code> ja <code>und 7<16?</code> ja, <code>positionLinks=1</code>
  <code>1<3?</code> ja <code>und 14<16?</code> ja, <code>positionLinks=2</code>
  <code>2<3?</code> ja <code>und 21<16?</code> nein
  <code>2>0?</code> ja <code>und 21>=16?</code> ja; <code>positionRechts=1</code>
  <code>1>0?</code> ja <code>und 14>=16?</code> nein; <code>positionLinks<positionRechts</code>: <code>2<1?</code>
nein //also kein Tausch
<code>2<1?</code> nein //äußere Schleife zu Ende
<code>21>16?</code> ja, <code>testArray=tauscheElemente(testArray,2,3)</code>:
<table><tr><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>7</td><td>14</td><td>16</td><td>21</td></tr></table>
<code>return 2</code>
Ausgabe <code>2</code>

<code>QuicksortArray.quicksort(testArray,0,1)</code>
<code>start=0, ende=1, 0<1?</code> ja
<code>q=teileArray(testArray,0,1)</code>
  <code>positionLinks=0, positionRechts=0</code>, <code>m=0</code>
<code>testArray=tauscheElemente(testArray,0,0)</code>:
<table><tr><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>7</td><td>14</td><td>16</td><td>21</td></tr></table>
<code>pivotElement=21</code>
solange <code>positionLinks<positionRechts</code>: <code>0<0?</code> nein
<code>7>21?</code> nein
<code>return 0</code>
Ausgabe <code>0</code>
bestimmen
angeben</td><td>

6</td><td>2</td><td></td></tr>
</table>

Aufg.	erwartete Leistungen	BE		
		I	II	III

1.6.3	Implementieren
	`private static int teileKarten(List<Karte> karten,`
	`    int start, int ende) {`
	`  int positionLinks = start;`
	`  int positionRechts = ende - 1;`
	`  int m = (ende + start) / 2;`
	`  karten = tauscheKarte(karten, m, ende);`
	`  Karte pivot = karten.get(ende);`
	`    while(positionLinks < positionRechts) {`
	`      while(karten.get(positionLinks).compareTo(pivot) < 0`
	`        && positionLinks < ende)`
	`          positionLinks++;`
	`      while(karten.get(positionRechts).compareTo(pivot) >=`
	`0`
	`        && positionRechts > start)`
	`          positionRechts--;`
	`      if(positionLinks < positionRechts) {`
	`         karten = tauscheKarte(karten,positionLinks,`
	`           positionRechts);`
	`         }`
	`      }`
	`    if(karten.get(positionLinks).compareTo(pivot) > 0) {`
	`       karten = tauscheKarte(karten, positionLinks, ende);`
	`    }`
	`    return positionLinks;`
	`  }`
	`  private static List<Karte>`
	`    tauscheKarte(List<Karte> karten, int a, int b) {`
	`    Karte temp = karten.get(a);`
	`    karten.set(a, karten.get(b));`
	`    karten.set(b, temp);`
	`    return karten;`
	`  }`
			4	3

	Summe 60	17	25	18

Aufg.	erwartete Leistungen	BE I	BE II	BE III
2.1	beschreiben Das vorliegende ERM besteht aus vier Entitätstypen, deren Bezeichnungen in den Rechtecken abzulesen sind, einer davon ist Spieler. Mit den drei Rauten sind die vorhandenen Beziehungen (Relationships) dargestellt. Zum Beispiel kann jeder Spieler zu einem Zweierteam gehören, und jedes Zweierteam besteht aus genau zwei Spielern. Die zugehörigen Kardinalitäten sind in min-max-Form in eckige Klammern gesetzt. Die Attribute stehen in ellipsenförmigen Notationselementen und gehören meistens zu Entitätstypen wie zum Beispiel der Preis zu einem Exemplar Kartensatz. Im ERM muss jeder Entitätstyp einen Primärschlüssel haben, den man am unterstrichenen Attribut erkennt: `cSpieliD` ist der Primärschlüssel von `Canastaspiel`. Attribute können auch zu einer Beziehung gehören wie hier `punktezahl` zu `nimmt teil`.	3		
	angeben Ein ER-Modell ermöglicht eine übersichtliche Darstellung des konzeptionellen Entwurfs einer Datenbank. Es wird zur grafischen Veranschaulichung des geplanten Abstraktionskonzepts eingesetzt, nachdem die Anforderungsanalyse abgeschlossen ist.	1		

2.2	überführen
	`Spieler(sID, name, vorname, gebDat)`
	`Team(teamID, bewertMod, sNr1#, sNr2#)`
	`NimmtTeil(teamID#, cSpielID#, punktezahl)`
	`Canastaspiel(cSpielID, datum, startZeit, ort, kID#)`
	`Kartensatz(kID, bezeichnung, herkunft, kaufDat, preis)`
		5		

	begründen Aus jeder der vier Entitätstypen entsteht eine Relation. Zu jedem Team gehören genau zwei Personen, und jede Person kann mehreren Teams angehören. Diese Beziehung ist zwar eine n:m-Beziehung und wird laut Transformationsregel in eine eigene Relation überführt. Da es sich aber hier immer um genau zwei Spieler handelt, ist es übersichtlicher, in der Tabelle `Team` jeweils einen FK für jeden Spieler aufzunehmen. Die n:m-Beziehung `nimmt teil` wird als zusätzliche Relation überführt, weil bis zu drei Zweierteams an einer Canastarunde teilnehmen können und das Attribut `punktezahl` von der Beziehung `nimmtTeil` abhängt. Die PK der beteiligten Tabellen werden FK in der Beziehungsmengenrelation. Die 1:n-Beziehung `wird benötigt` wird laut Transformationsregel durch den Fremdschlüssel mit der Karten-ID in der Relation `Canasta` realisiert. Die Nicht-Schlüsselattribute der Relationen sind vollständig vom jeweiligen PK abhängig und es existieren keine transitiven Abhängigkeiten, so dass die 3. Normalform eingehalten wird.		3	

2.3.1	Entwickeln
	`INSERT INTO Spieler VALUES (57, 'Schmidt', 'Petra',`
	`'2001-07-19');`
	`INSERT INTO Team (bewertMod, sNr1, sNr2)`
	`VALUES (false, (SELECT sID FROM Spieler`
	`WHERE name LIKE 'Treusch'`
	`AND vorname LIKE 'Astrid'),`
	`57);`
			2	3

Aufg.	erwartete Leistungen	BE I	BE II	BE III

2.3.2	entwickeln
	`UPDATE Canastaspiel SET kID =13`
	`WHERE datum > NOW() AND kID = 1;`
	`UPDATE Canastaspiel SET kID = NULL`
	`WHERE kID = 1;`
	`DELETE FROM Kartensatz WHERE kID = 1;`
			4	1

2.3.3	implementieren
	`SELECT teamID, cSpielID, punktezahl`
	`FROM Nimmtteil`
	`WHERE punktezahl >= 5000;`
			2	

2.3.4	entwickeln
	`SELECT c.kID, COUNT(c.kID) AS Anzahl`
	`FROM Canastaspiel c, Kartensatz e`
	`WHERE c.kID = e.kID`
	`GROUP BY c.kID`
	`HAVING COUNT(c.kID) > 100`
	`ORDER BY Anzahl DESC;`
			2	3

2.3.5	entwickeln
	`SELECT sp.sID, sp.vorname, sp.name, SUM(n.punktezahl) AS Gesamtpunkte`
	`FROM Spieler sp, Team z, Nimmtteil n`
	`WHERE z.teamID = n.teamID AND (sp.sID = z.sID1 OR sp.sID = z.sID2) AND z.bewertMod = true`
	`GROUP BY sp.sID;`
			3	1

2.4	entwickeln, zeichnen entwickeln zeichnen	 4	1	2
	Summe 40	**13**	**17**	**10**

III Bewertung und Beurteilung

Die Bewertung und Beurteilung erfolgt unter Beachtung der nachfolgenden Vorgaben nach § 33 der Oberstufen- und Abiturverordnung (OAVO) in der jeweils geltenden Fassung. Bei der Bewertung und Beurteilung der sprachlichen Richtigkeit in der deutschen Sprache sind die Bestimmungen des § 9 Abs. 12 Satz 3 OAVO in Verbindung mit Anlage 9b anzuwenden.

Bei der Bewertung und Beurteilung der Übersetzungsleistung in den Fächern Latein und Altgriechisch sind die Bestimmungen des § 9 Abs. 14 OAVO in Verbindung mit Anlage 9c anzuwenden.

Der Fehlerindex ist nach Anlage 9b zu § 9 Abs. 12 OAVO zu berechnen. Für die Ermittlung der Punkte nach Anlage 9a zu § 9 Abs. 12 OAVO sowie Anlage 9c zu § 9 Abs. 14 OAVO wird jeweils der ganzzahlige nicht gerundete Prozentsatz bzw. Fehlerindex zugrunde gelegt.

Für die Bewertung in den modernen Fremdsprachen ist der „Erlass zur Bewertung und Beurteilung von schriftlichen Arbeiten in allen Grund- und Leistungskursen der neu beginnenden und fortgeführten modernen Fremdsprachen in der gymnasialen Oberstufe, dem beruflichen Gymnasium, dem Abendgymnasium und dem Hessenkolleg“ vom 7. August 2020 (ABl. S. 519) zugrunde zu legen. Demnach erfolgt die Bewertung und Beurteilung mit der Maßgabe, dass lediglich bei der Ermittlung des Prüfungsergebnisses (Note) aus Prüfungsteil 1 und 2 gerundet wird.

Darüber hinaus sind die Vorgaben der Erlasse „Hinweise zur Vorbereitung auf die schriftlichen Abiturprüfungen (Abiturerlass)“, „Hinweise zur Vorbereitung auf die schriftlichen Abiturprüfungen im beruflichen Gymnasium (fachrichtungs-/ schwerpunktbezogene Fächer) (Abiturerlass BG)“ und „Durchführungsbestimmungen zum Landesabitur“ in der für den Abiturjahrgang geltenden Fassung zu beachten.

Als Kriterien für die Bewertung und Beurteilung dienen unter Beachtung der Zielsetzung der gymnasialen Oberstufe nach § 1 Abs. 2 OAVO neben dem Inhaltlichen auch die in den Kerncurricula genannten überfachlichen Kompetenzen, insbesondere die Sprachkompetenz und Wissenschaftspropädeutik; dies zeigt sich u.a. in qualitativen Merkmalen wie Strukturierung, Differenziertheit, (fach-)sprachlicher Gestaltung und Schlüssigkeit der Argumentation.

Im Fach Praktische Informatik besteht die Prüfungsleistung aus der Bearbeitung von zwei Aufgabenmodulen, wofür insgesamt maximal 100 BE vergeben werden können. Ein Prüfungsergebnis von **5 Punkten (ausreichend)** setzt voraus, dass mindestens 45% der zu vergebenden BE erreicht werden. Ein Prüfungsergebnis von **11 Punkten (gut)** setzt voraus, dass mindestens 75% der zu vergebenden BE erreicht werden.

Gewichtung der Aufgaben und Zuordnung der Bewertungseinheiten zu den Anforderungsbereichen

Aufgabe	**Bewertungseinheiten in den Anforderungsbereichen**			**Summe**
	AFB I	**AFB II**	**AFB III**	
1	17	25	18	**60**
2	13	17	10	**40**
Summe	**30**	**42**	**28**	**100**

Die auf die Anforderungsbereiche verteilten Bewertungseinheiten innerhalb der Aufgaben sind als Richtwerte zu verstehen.